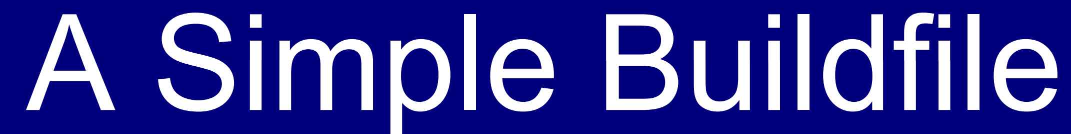


A Simple Buildfile

Packing and Running a Basic Java Application

Sample Content

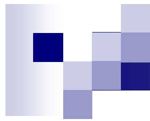
© Garth Gilmour 2008

garth@ggilmour.com

Examining A Simple Buildfile

- As previously noted Ant buildfiles are XML documents
  - All the content goes inside '<project>' tags
  - In XML terminology this is the 'document element'
- Most Ant buildfiles have a very simple structure
  - First come properties that represent 'magic numbers'
    - The names of directories, archives, users, servers etc…
  - Second come more complex data types
    - These represent sets of files and directories
  - Third and finally come the targets
    - These represent the things we want Ant to do
    - Typical names for targets are 'init', 'clean', 'run' etc…

© Garth Gilmour 2008

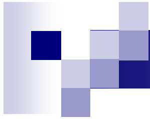


```
<project  name="Example Ant Script" default="runApp" basedir=".">

    <!-- Set up the directories used in the build -->
    <property name="build.src" value="..\src"/>
    <property name="build.bin" value="..\bin"/>
    <property name="build.deploy" value="..\deploy"/>
    <!-- Set up the name of the jar file to be built -->
    <property name="jarName" value="myapp.jar"/>
    <!-- Set up the path to the jar file -->
    <property name="pathToJar" value="${build.deploy}\${jarName}"/>
    <!-- Set up the name of the class which starts the app -->
    <property name="launcherClassName"
                    value="tools.ant.basic.Demo"/>

    <!-- Set up a data type for the classpath -->
    <path id="myClasspath">
        <pathelement location="${build.bin}"/>
        <pathelement location="${build.deploy}"/>
    </path>
```

Properties

Path

© Garth Gilmour 2008



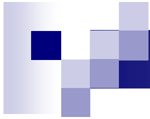
Target with no dependencies

```
<!-- Use the javac task to compile the bean classes -->
<target name="compileClasses">
      <javac srcdir="${build.src}" destdir="${build.bin}"
                  includes="**/*.java">
            <classpath refid="myClasspath"/>
      </javac>
</target>
```

Target which depends on ‘compileClasses’

```
<!-- Use the jar task to build the archive -->
<target name="buildJar" depends="compileClasses">
      <jar jarfile="${jarName}">
            <fileset dir="${build.bin}">
                  <include name="**\*.class"/>
            </fileset>
            <manifest>
                  <attribute name="Main-Class"
                                    value="${launcherClassName}"/>
            </manifest>
      </jar>
</target>
```

© Garth Gilmour 2008



Target which depends on target ‘buildJar’

Target which depends on target ‘runApp’

```
<!-- Move the finished archive to the deployment directory -->
<target name="deploy" depends="buildJar">
      <move file="${jarName}" todir="${build.deploy}"/>
</target>

<!-- Run the application as an executable JAR -->
<target name="runApp" depends="deploy">
      <java fork="true" jar="${pathToJar}">
            <arg value="20"/>
      </java>
</target>

</project>
```

© Garth Gilmour 2008

Examining A Simple Buildfile

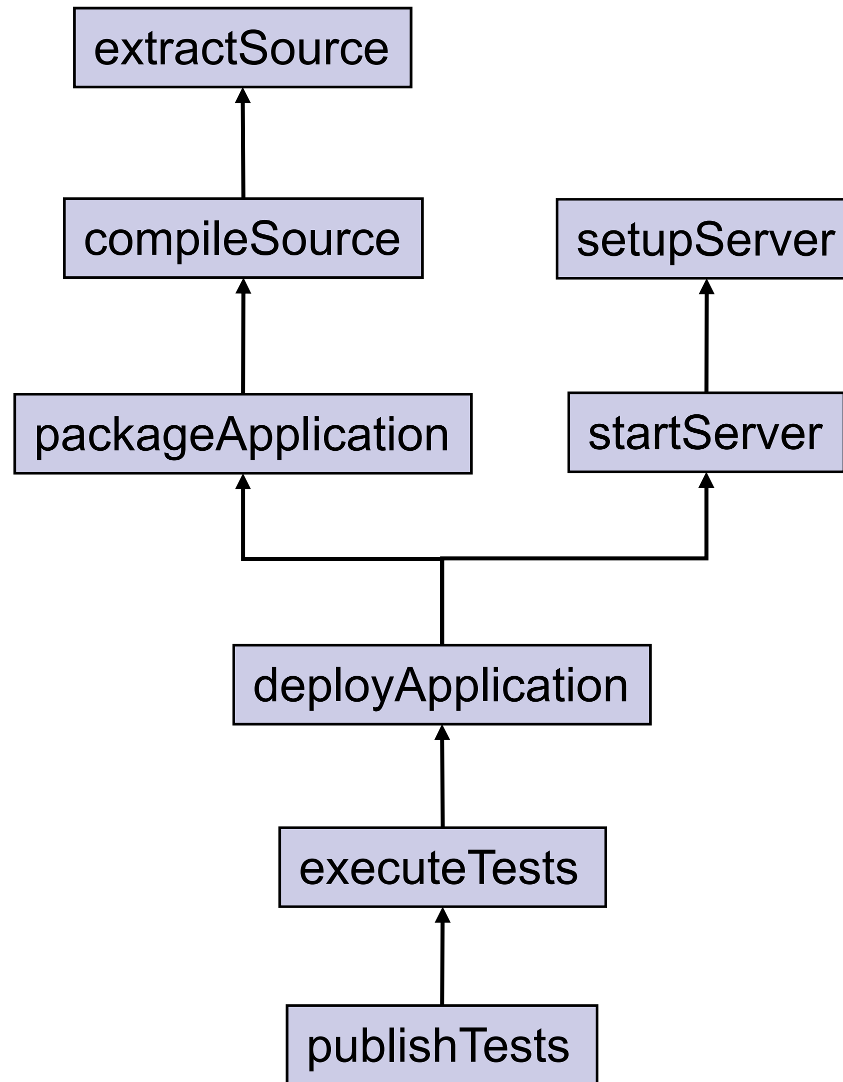
- Most of an Ant buildfile is made up of targets
  - Each target contains one or more tasks
  - Tasks are (mostly) supplied by Ant
    - They represent actions it can carry out
- Targets have dependencies on one another
  - If target ‘A’ is dependent on ‘B’ and ‘C’ then before ‘A’ can be executed Ant will trigger the ‘B’ and ‘C’ targets
  - E.g. the ‘run’ target depends on ‘compile’
    - Which in turn depends on ‘extractSource’
- When you run a buildfile the default target is called
  - This is the one specified in the ‘default’ attribute of ‘project’

© Garth Gilmour 2008

Examining A Simple Buildfile

- When Ant loads a buildfile it creates a dependency tree
  - This links each target to any others it is dependent on
  - If it detects circular dependencies the file will not run
- Each target will have a chain of dependencies
  - A dependency is only ever executed once
    - No matter how many times it appears in the chain
- Multiple dependencies are usually run from left to right
  - But this may be overridden by other relationships
  - E.g. given ‘<target name="A" depends="B,C">’ and ‘<target name="B" depends="C">’ the order will be C→B→A

© Garth Gilmour 2008



© Garth Gilmour 2008

Properties and Data Types

Representing Resources

© Garth Gilmour 2008

garth@ggilmour.com

Properties in Ant Buildfiles

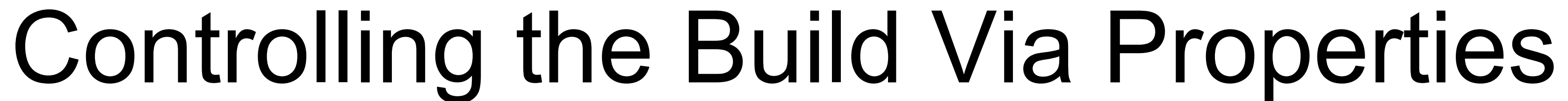
- Properties are the most common item in a buildfile
  - They encapsulate all the ‘magic strings’ that will be used later
    - Directory and file names, URL’s, archive names, passwords etc…
  - Ant provides some built in properties
    - The ‘ant.file’ property gives the path to the build file
    - The ‘ant.home’ property gives Ants installation directory
- The syntax for declaring properties is straightforward
  - E.g. ‘<property name=“build.test” value=“abc123”>’
- Properties are accessed via the ‘${ *name* }’ syntax
  - One property name can be used in the declaration of another
  - E.g. ‘<property name=“a” value=“${output.dir}/${jarName}”>’

© Garth Gilmour 2008

Loading Properties from Files

- Properties can be loaded from a file
  - When declaring a property you can say ‘file=“build.properties”’
  - The file should contain one or more ‘a=b’ pairs
    - These are exported as Ant properties
    - E.g. ‘tomcat.home=C:\\tomcat60’
- The values of properties are immutable
  - Once a property has been set its value cannot be changed
  - Properties elements later in the file cannot change the value
  - This allows settings in the file to override those in the build

© Garth Gilmour 2008



Controlling the Build Via Properties

- Targets can be conditionally run based on properties
  - E.g. ‘<target name=“abc” if=“*property_name*”>’
- Note that the value of the property does not matter
  - The condition depends only on whether the property is set
- The ‘fail’ task can be used to halt a build
  - A property name is specified via ‘if’ or ‘unless’
- The import of files can be made conditional
  - The ‘include’ and ‘exclude’ taks used in patternsets (see later) have optional ‘if’ and ‘unless’ attributes

© Garth Gilmour 2008

Data Types in Ant

- Ant provides a set of data types
  - Represented as classes within the implementation
- These are used to represent complex information
  - Typically lists of file paths, files and directories
  - That are to be used by tasks within targets
- The ‘patternset’ is probably the most fundamental
  - It specifies a set of file locations via nested elements
- Most of the other data types have an ‘implicit patternset’
  - This means they provide the functionality of a ‘patternset’
  - Any content that is valid in a ‘patternset’ is valid in them

© Garth Gilmour 2008

Data Type	Description
Argument	Used to pass arguments to the apply, exec and java tasks
Environment	Used by the apply and exec tasks to pass environment variables
FileList	Builds a list of filenames. The files need not exist.
FileSet	Builds a set of files based on patterns. Files must exist
DirSet	Builds a set of directories based on patterns
PatternSet	Groups a series of patterns. Often used in FileSet
FilterSet	Groups a set of filters that can parse and replace tokens in files
Path	Often used to build a classpath. Can appear as an element or an attribute
Mapper	Maps one type of file to another in order to copy, rename and move groups of files

© Garth Gilmour 2008

Ant Data Types

- When specifying file paths wildcards can be used
  - ‘?’ matches a single character and ‘*’ zero or more
  - ‘**’ is ant-specific and means ‘in all nested directories’
    - E.g. ‘<include name="**/*.txt"/>’ would select text files in the current directory and all nested directories
- A trailing slash is interpreted as ‘**’
  - E.g. pattern ‘../lib/’ is the same as ‘../lib/**’
- Ant ignores platform specific characters
  - It does not care about the direction of the slash in file paths
  - The separator used in paths can be either ‘:’ or ‘;’

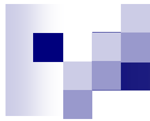
© Garth Gilmour 2008

Selectors in the FileSet Type

- In addition to the standard 'patternset' content a 'fileset' can contain what are known as 'selectors'
  - These select files based on criteria other than the filename
  - Selectors can be combined using '<or>', '<and>', '<none>' etc…

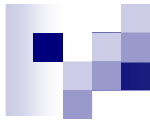
Selector	Description
depth	Selects files based on a depth in the directory tree
filename	Selects files that match a pattern (equivalent to include/exclude but can be used inside '<or>' etc…)
size	Selects files that are larger or smaller than a particular size
date	Selects files based on their modification date
contains	Selects files that contain a particular string

© Garth Gilmour 2008



```
<project name=“FileSet Data Type Demo" default="runDemo" basedir=".">
    <!-- including files via explicit patternset -->
    <fileset dir="..\input" id="fs1">
        <patternset>
            <include name="**/*"/>
        </patternset>
    </fileset>

    <!-- including and excluding files via explicit patternset -->
    <fileset dir="..\input" id="fs2">
        <patternset>
            <include name="**/*"/>
            <exclude name="**/*.txt"/>
        </patternset>
    </fileset>

    <!-- including files via implicit patternset -->
    <fileset dir="..\input" id="fs3">
        <include name="**/*"/>
    </fileset>
```

© Garth Gilmour 2008



```
<!-- including and excluding files via implicit patternset -->
<fileset dir="..\input" id="fs4">
    <include name="**/*"/>
    <exclude name="**/*.txt"/>
</fileset>

<!-- including files via filename selector -->
<fileset dir="..\input" id="fs5">
    <filename name="**/*"/>
</fileset>

<!-- including and excluding files via filename selector -->
<fileset dir="..\input" id="fs6">
    <filename name="**/*"/>
    <filename name="**/*.txt" negate="true"/>
</fileset>

<!-- including files via the depth selector -->
<fileset dir="..\input" id="fs7" includes="**/*">
    <depth max="1"/>
</fileset>
```

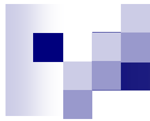
© Garth Gilmour 2008

```
<!-- including files via the contains selector -->
<fileset dir="..\input" id="fs8" includes="**/*">
      <contains text="--- marker ---"/>
</fileset>

<!-- including files via the size selector -->
<fileset dir="..\input" id="fs9" includes="**/*">
      <size value="100" when="more"/> <!-- default size is bytes -->
</fileset>

<!-- combining selectors using or -->
<fileset dir="..\input" id="fs10" includes="**/*">
      <or>
            <size value="100" when="more"/> <contains text="--- marker ---"/>
      </or>
</fileset>

<!-- combining selectors using and -->
<fileset dir="..\input" id="fs11" includes="**/*">
      <and>
            <size value="100" when="more"/> <contains text="--- marker ---"/>
      </and>
</fileset>
```

© Garth Gilmour 2008

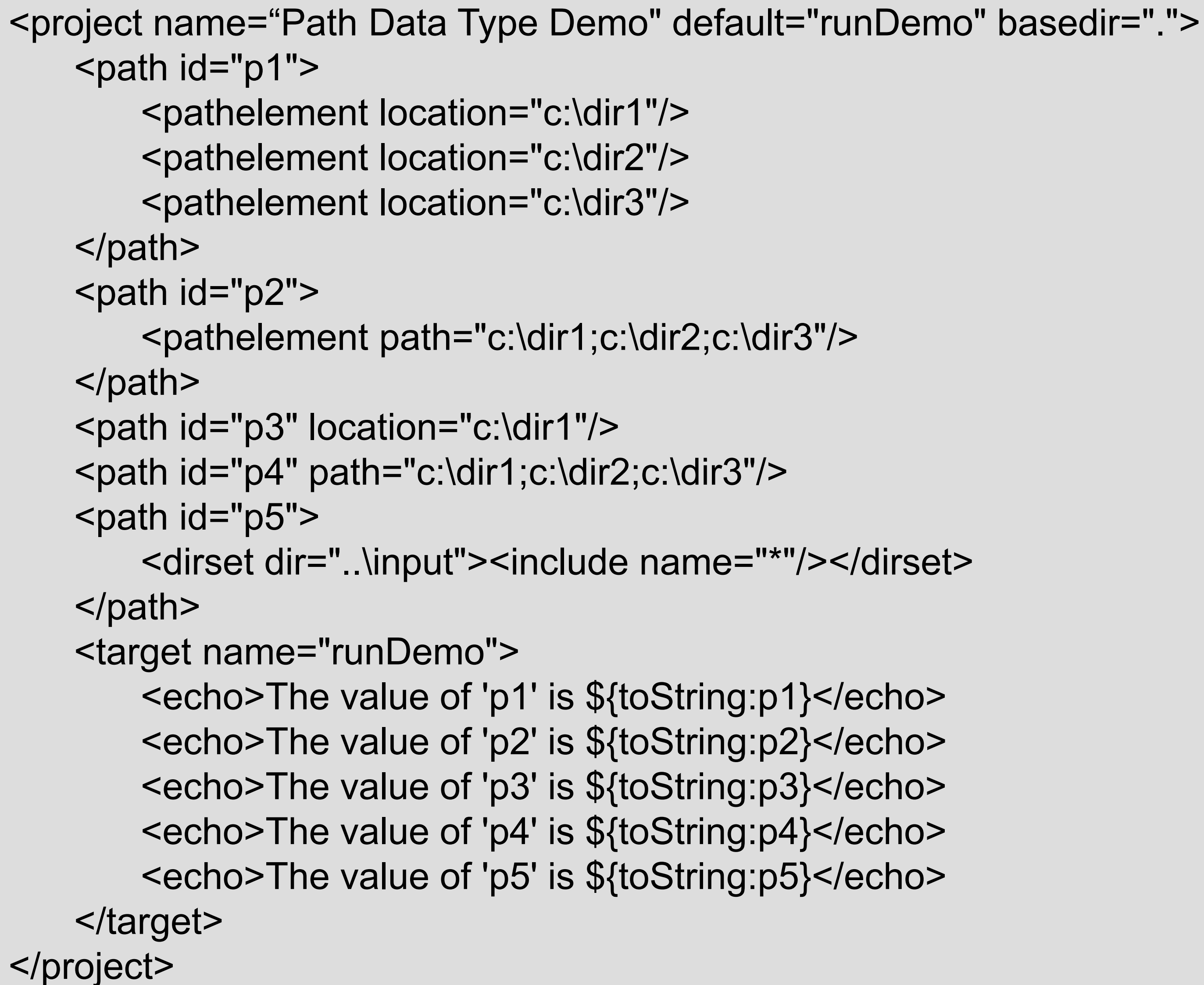
```
    <!-- combining selectors using none -->
    <fileset dir="..\input" id="fs12" includes="**/*">
        <none>
            <size value="100" when="more"/> <contains text="--- marker ---"/>
        </none>
    </fileset>

    <target name="runDemo">
        <echo>The value of 'fs1' is ${toString:fs1}</echo>
        <echo>The value of 'fs2' is ${toString:fs2}</echo>
        <echo>The value of 'fs3' is ${toString:fs3}</echo>
        <echo>The value of 'fs4' is ${toString:fs4}</echo>
        <echo>The value of 'fs5' is ${toString:fs5}</echo>
        <echo>The value of 'fs6' is ${toString:fs6}</echo>
        <echo>The value of 'fs7' is ${toString:fs7}</echo>
        <echo>The value of 'fs8' is ${toString:fs8}</echo>
        <echo>The value of 'fs9' is ${toString:fs9}</echo>
        <echo>The value of 'fs10' is ${toString:fs10}</echo>
        <echo>The value of 'fs11' is ${toString:fs11}</echo>
        <echo>The value of 'fs12' is ${toString:fs12}</echo>
    </target>
</project>
```

© Garth Gilmour 2008



```
<project name="DirSet Data Type Demo" default="runDemo" basedir=".">
    <!-- A dirset is similar to a fileset but only selects directories -->
    <dirset dir="..\input" id="ds1">
        <include name="*"/>
    </dirset>
    <dirset dir="..\input" id="ds2">
        <include name="*"/>
        <exclude name="*3"/>
    </dirset>
    <dirset dir="..\input" id="ds3">
        <include name="**"/>
    </dirset>

    <target name="runDemo">
        <echo>The value of 'ds1' is ${toString:ds1}</echo>
        <echo>The value of 'ds2' is ${toString:ds2}</echo>
        <echo>The value of 'ds3' is ${toString:ds3}</echo>
    </target>
</project>
```

© Garth Gilmour 2008



```
<project name="Path Data Type Demo" default="runDemo" basedir=".">
    <path id="p1">
        <pathelement location="c:\dir1"/>
        <pathelement location="c:\dir2"/>
        <pathelement location="c:\dir3"/>
    </path>
    <path id="p2">
        <pathelement path="c:\dir1;c:\dir2;c:\dir3"/>
    </path>
    <path id="p3" location="c:\dir1"/>
    <path id="p4" path="c:\dir1;c:\dir2;c:\dir3"/>
    <path id="p5">
        <dirset dir="..\input"><include name="*"/></dirset>
    </path>
    <target name="runDemo">
        <echo>The value of 'p1' is ${toString:p1}</echo>
        <echo>The value of 'p2' is ${toString:p2}</echo>
        <echo>The value of 'p3' is ${toString:p3}</echo>
        <echo>The value of 'p4' is ${toString:p4}</echo>
        <echo>The value of 'p5' is ${toString:p5}</echo>
    </target>
</project>
```

© Garth Gilmour 2008

Using Core, Optional and External Tasks

© Garth Gilmour 2008

garth@ggilmour.com

Tasks in Ant

- There are three kinds of Ant tasks
  - Core tasks come with Ant and are self contained
    - Their only dependencies are on standard JSE libraries
  - Optional tasks come with Ant but have external requirements
    - They require that other software be installed on the machine
    - Such as version control systems, JEE servers and testing tools
  - Custom tasks are provided by other organisations
    - In order to make their products easy to use in build scripts
- We will not be examining every task in detail
  - Instead we will focus on those that are complex and/or useful
  - The Ant manual documents each task in detail

© Garth Gilmour 2008

Task	Description
apply, exec	Execute a system command
copy	Copies files and directories
move	Moves files and directories
mkdir	Creates directories
javac	Compiles source code
rmic	Generates RMI stubs
cvs	Executes CVS commands
jar	Builds a Java Archive
war	Builds a Web Archive
ear	Builds an Enterprise Archive
style	Performs an XSLT transformation
antstructure	Generates a DTD for a buildfile

© Garth Gilmour 2008

Example Optional Tasks

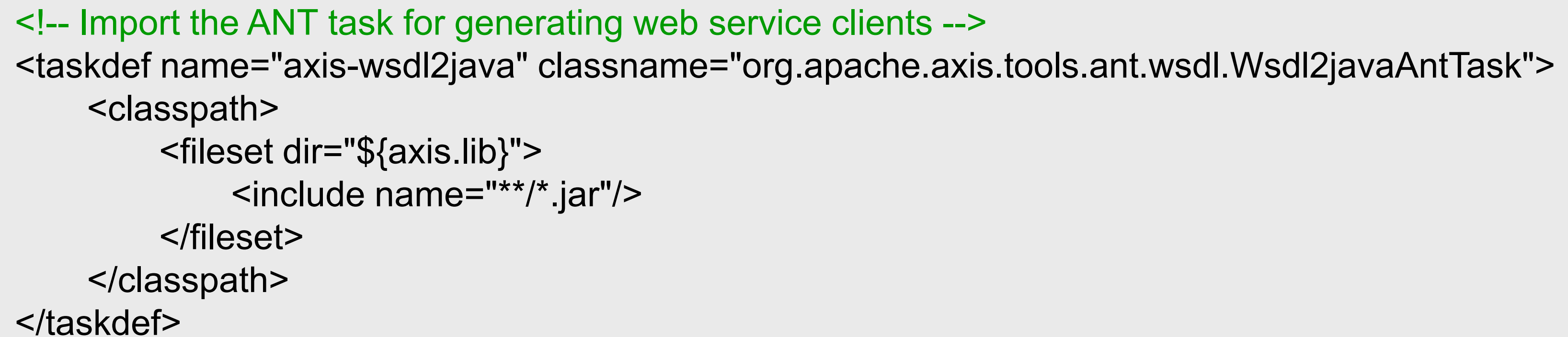
Task	Description
junit	Run JUnit tests
junitreport	Reformat JUnit test results
cc***	Tasks for working with ClearCase
vss***	Tasks for working with SourceSafe
p4***	Tasks for working with Perforce
csc	Run the C# compiler
ilasm	Generate .NET assemblies
telnet	Open a telnet session with a server
rpm	Builds an archive for distribution on Linux
antlr	Uses ANTLR to build a language interpreter

© Garth Gilmour 2008

Importing External Tasks

- External tasks can be used via ‘taskdef’ and ‘typedef’
  - These can be paced at the top level or nested inside targets
- The ‘taskdef’ target has three main attributes
  - The ‘name’ attribute gives the identifier that will be used
  - The ‘classname’ attribute specifies the implementing class
  - The ‘classpath’ attribute specifies the library holding the class
- There are other ways of specifying external tasks
  - When creating an ‘antlib’ extension library the mapping of task names to classes can be specified via XML or properties files

© Garth Gilmour 2008



```
<!-- Import the ANT task for generating web service clients -->
<taskdef name="axis-wsdl2java" classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask">
	<classpath>
		<fileset dir="${axis.lib}">
			<include name="**/*.jar"/>
		</fileset>
	</classpath>
</taskdef>
```

```
<!-- Import the Sun task for generating web service implementations -->
<taskdef name="wsgen" classname="com.sun.tools.ws.ant.WsGen">
	<classpath refid="myClasspath"/>
</taskdef>
```

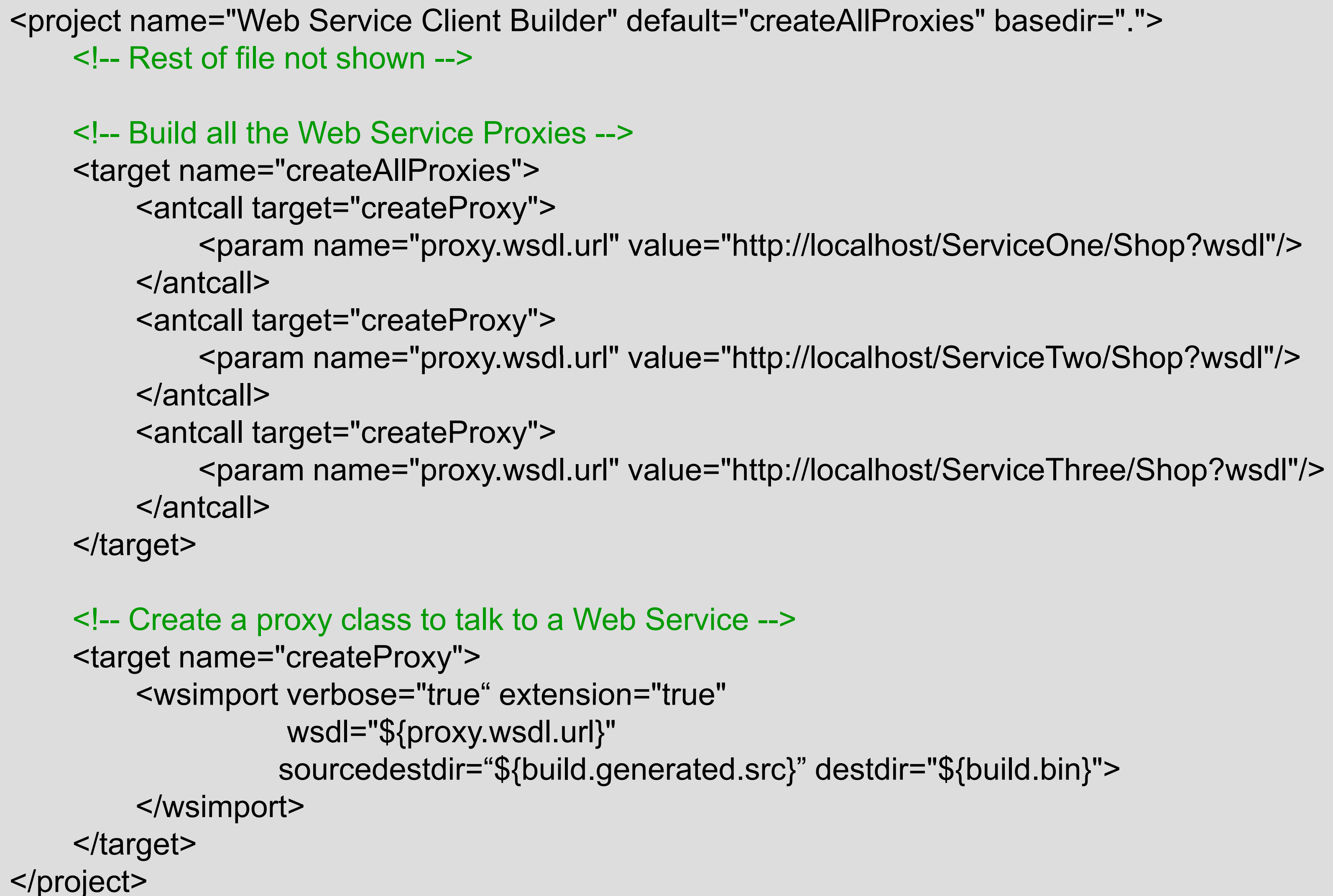
```
<!-- Import the BEA WebLogic task for generating web service implementations -->
<taskdef name="servicegen"
		classname="weblogic.ant.taskdefs.webservices.servicegen.ServiceGenTask">
	<classpath>
		<pathelement path="${weblogic.lib}\weblogic.jar"/>
		<pathelement path="${weblogic.lib}\webservices.jar"/>
	</classpath>
</taskdef>
```

© Garth Gilmour 2008

Calling Other Tasks

- The ‘antcall’ task lets you call another target
  - In a manner analogous to a normal function call
- Additional properties can be passed to the target
  - Via one or more nested ‘param’ elements
- Any dependencies of the called target are executed
  - They will be able to see any additional parameters
- This task is very useful when you need to run the same process multiple times for different inputs
  - Such as generating clients for different EJB’s or Web Services

© Garth Gilmour 2008



```
<project name="Web Service Client Builder" default="createAllProxies" basedir=".">
	<!-- Rest of file not shown -->

	<!-- Build all the Web Service Proxies -->
	<target name="createAllProxies">
		<antcall target="createProxy">
			<param name="proxy.wsdl.url" value="http://localhost/ServiceOne/Shop?wsdl"/>
		</antcall>
		<antcall target="createProxy">
			<param name="proxy.wsdl.url" value="http://localhost/ServiceTwo/Shop?wsdl"/>
		</antcall>
		<antcall target="createProxy">
			<param name="proxy.wsdl.url" value="http://localhost/ServiceThree/Shop?wsdl"/>
		</antcall>
	</target>

	<!-- Create a proxy class to talk to a Web Service -->
	<target name="createProxy">
		<wsimport verbose="true“ extension="true"
				 wsdl="${proxy.wsdl.url}"
				sourcedestdir=“${build.generated.src}” destdir="${build.bin}">
		</wsimport>
	</target>
</project>
```

© Garth Gilmour 2008

Running Multiple Buildfiles

- The ‘ant’ tasks executes another buildfile
  - The ‘dir’ attribute specifies the location of the file
    - This defaults to the base directory of the current project
  - The ‘antfile’ attribute specifies the name of the file
    - This defaults to ‘build.xml’
  - The ‘target’ attribute specifies the target to run
    - By default this is the default target in the project being called
- This is invaluable when packaging large applications
  - E.g. build multiple libraries into JAR files, place the JAR’s inside one or more WAR’s and then insert the WAR’s into an EAR

© Garth Gilmour 2008

Running Multiple Buildfiles

```
<project name="Do Everything" default="deployAllWebApps" basedir=".">
    <!-- Run the ANT build files for all the Web Application projects -->
    <target name="deployAllWebApps">
        <ant dir="./WebAppOne" antfile="./build.xml" target="deployToTomcat"/>
        <ant dir="./WebAppTwo" antfile="./build.xml" target="deployToTomcat"/>
        <ant dir="./WebAppThree" antfile="./build.xml" target="deployToTomcat"/>
        <ant dir="./WebAppFour" antfile="./build.xml" target="deployToTomcat"/>
        <ant dir="./WebAppFive" antfile="./build.xml" target="deployToTomcat"/>
        <ant dir="./WebAppSix" antfile="./build.xml" target="deployToTomcat"/>
    </target>
</project>
```

© Garth Gilmour 2008

Copying Files

- The ‘copy’ task is extremely versatile
  - It can be used to rename, copy or transform files
- Transformations are applied via ‘mapper’ elements
  - The ‘type’ attribute of the mapper specifies the transformation

Value of ‘type’	Description
flatten	Copies a file but not any nested directory names.
glob / regexp	Copies files based on a pattern, the name of the copied file can be altered. The glob type only uses ‘*’ whereas regexp uses full regular expressions.
package/unpackage	Replaces nested directories with dots in the file name, or vice versa
composite	Nests multiple mappers, each of which is allied to each file selected
chained	Nests multiple mappers, files are passed through each mapper in turn

© Garth Gilmour 2008

Compiling Java Code

- The ‘javac’ task runs the Java compiler
  - You can use the ‘compiler’ attribute to specify which compiler
- A file will be compiled when:
  - There is no matching ‘.class’ file in the output directory
  - The matching ‘.class’ file is older than the source file
    - Time related errors can occur when moving files over the network
- The source files must be laid out in a package hierarchy
  - Modern IDE’s all automatically organise ‘.java’ files in this way
  - Continuous compilation in Eclipse can make ‘javac’ redundant

```
<javac srcdir="${build.src}" destdir="${build.bin}">
      <classpath refid="buildClasspath" />
</javac>
```

© Garth Gilmour 2008

Building Archives

- The ‘zip’ and ‘jar’ tasks are closely related
  - In the implementation ‘jar’ is based on top of ‘zip’
  - Java developers will create JAR’s more other than ZIP’s
- File information can be specified in two ways
  - Via the tasks attributes or via nested patternset content
  - The ‘zipfileset’ data type lets you insert content directly

```
<jar jarfile="${jarName}">
        <fileset dir="${build.bin}">
                <include name="**\*.class"/>
        </fileset>
        <fileset dir="${build.meta}\..">
                <include name="META-INF\*.xml"/>
        </fileset>
</jar>
```

```
<jar jarfile="${jarName}">
        <fileset dir="${build.bin}">
                <include name="**\*.class"/>
        </fileset>
        <manifest>
                <attribute name="Main-Class"
                  value="${launcherClassName}"/>
        </manifest>
</jar>
```

© Garth Gilmour 2008

Building Archives

- There are two tasks for building JEE components
  - The ‘war’ task builds a Web Archive
    - In order to package up a Web Application
  - The ‘ear’ task builds an Enterprise Archive
    - In order to package up an Enterprise Application
- These tasks add convenience features to ‘jar’
  - They are aware of where items need to be inserted
  - E.g. the ‘webxml’ attribute places a file into ‘WEB-INF’
- Note that tasks also exist for unpacking archives
  - These are ‘unzip’, ‘unjar’ and ‘unwar’

© Garth Gilmour 2008

Running Java Classes

- The ‘java’ task runs a class within the VM
  - By default this is the VM running the buildfile
  - The ‘fork’ attribute starts another VM in a separate process
    - Executable JAR files must run in a separate VM
- The classpath can be set via an attribute or element
  - If the classpath will be reused it should be declared separately
    - Several classpaths may be needed at different points in the build
  - Remember the CLASSPATH environment variable can be disabled by running Ant with the ‘-noclasspath’ option
- Both VM and program arguments can be specified
  - Via nested ‘arg’ and ‘jvmarg’ elements

© Garth Gilmour 2008

```
<java fork="true" jar="${pathToJar}">
    <arg value="20"/>
</java>
```

```
<java fork="true" classname="${launcherClassName}">
    <arg value="${container.type}"/>
    <arg value="${jms.topic}"/>
    <classpath refid="classpath"/>
</java>
```

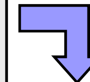
```
<!-- Run the weblogic ejb compiler in a separate VM -->
<java classname="weblogic.appc" fork="true">
    <arg line=" -output ${beaJarName} ${jarName} "/>
    <classpath>
        <pathelement path="${weblogic.lib}\weblogic.jar"/>
        <pathelement path="${weblogic.jvm}\lib\tools.jar"/>
    </classpath>
</java>
```

© Garth Gilmour 2008

Running Java Classes

- There is a very important issue when running JAR's
  - The only way to add values to the classpath is by setting the 'Class-Path' entry in the manifest within the JAR file
  - This contains one or more relative paths to other JAR files

```
<jar jarfile="${jarName}">
	<fileset dir="${build.bin}">
		<include name="**\*.class" />
	</fileset>
	<fileset dir="${build.config}">
		<include name="**\*.xml" />
	</fileset>
	<manifest>
		<attribute name="Main-Class" value="${launcherClassName}" />
		<attribute name="Class-Path" value="${jarClasspath}" />
	</manifest>
</jar>
```



```
<java fork="true" jar="${pathToJar}">
```

© Garth Gilmour 2008

ANT And Unit Testing

- JUnit can be used in build files via the ‘junit’ task
  - It runs test cases and saves the results into files
  - The JUnit tests can test code directly
    - Or use extensions like HttpUnit to test web components
- Results can be displayed as plain text
  - The ‘printsummary’ attribute produces a terse report
  - Adding a ‘formatter’ element generates a more verbose report
    - Which can be sent to the console or a file
- The results can be converted to HTML
  - The ‘junitreport’ task takes XML output from the ‘junit’ task and runs an XSLT stylesheet to convert the results

© Garth Gilmour 2008

ANT and Unit Testing

```
<!-- Run JUnit test classes. An xml report will be placed in 'build.reports‘ -->
<target name="runTests" depends="compileClasses">
    <junit haltonerror="true" haltonfailure="true" printsummary="on">
        <classpath refid="classpath"/>
        <formatter type="xml" usefile="true"/>
        <test name="${servletTestClassName}" todir="${build.reports}"/>
        <test name="${jspTestClassName}" todir="${build.reports}"/>
    </junit>
</target>
<!-- Combine the JUnit reports and transform the result into HTML -->
<target name="transformReports" depends="runTests">
    <junitreport todir="${build.reports.html}">
        <fileset dir="${build.reports}">
            <include name="*.xml"/>
        </fileset>
        <report format="frames" todir="${build.reports.html}"/>
    </junitreport>
</target>
```

© Garth Gilmour 2008